An Adaptable Benchmark for MPFS Performance Testing

by

Yubing Wang

A Thesis

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

in partial fulfillment of the requirements for the

Degree of Master of Science

in

Computer Science

by

May 2002

APPROVED:

_____

Professor Mark Claypool, Thesis Advisor

_____

Professor Bob Kinicki, Thesis Reader

_____

Professor Micha Hofri, Head of Department

Abstract

Multiplex File System (MPFS) is a new network file access protocol that allows heterogeneous hosts to access the shared data. The MPFS split data-metadata architecture demands new ways to evaluate performance. Current I/O benchmarks are not suitable to measure the performance of MPFS because they mainly measure the performance of NFS or CIFS. We develop a new benchmark that can measure the performance of MPFS for a variety of applications. Our benchmark uses a series of application groups to mimic the real-world applications that use MPFS. The mix of these application groups generates the workload to stress the MPFS system.

Based on our MPFS benchmark, we develop a *MPFS Performance Monitor for EMC Celerra File Server* (*CFSPM*). CFSPM is a benchmark program to measure the performance of Celerra File Server under MPFS protocol with a web-based interface. A series of MPFS performance data colleted from CFSPM is presented and analyzed in the thesis. We find that the CFSPM benchmark does meet the criteria we apply to the ideal MPFS benchmark.

Acknowledgments

I would like to thank my advisor, Professor Mark Claypool, for the guidance he provided during this thesis. I would also like to thank Professor Bob Kinicki for reading this thesis and providing valuable feedbacks. My thanks also go to EMC Corporation for supporting me to complete this work and my EMC colleague Zheng Zuo for his kindly support.

Contents

List of Figures

List of Tables

Chapter 1 Introduction

Increasingly, enterprises require reliable and high-speed access from multiple computers to shared data, with the ability to gracefully manage rapid growth of storage sizes and speeds. NFS (typically on UNIX systems) and CIFS (typically on Windows NT systems) are widely used methods to share files among multiple computers. They provide a logically shared view of file systems, allowing remote files to appear as if they were local files and allowing multiple computers to share the files.

Unfortunately, accessing remote files over networks can be significantly slower than accessing local files. The network hardware and software introduces delays that tend to significantly lower the transaction rates (I/Os per second) and the bandwidth (megabytes per second). These delays are difficult to avoid in the client-server architecture of network-based file systems. Network bandwidths impose an upper limit on the bandwidth of most existing shared file systems. The layers of network protocols and server software also tend to limit the bandwidth rates. A shared-file server can be a bottleneck of performance for multiple clients whose requests and data transfers must pass through the centralized file server.

In response to the demands for high-speed and highly scalable storage-to-server and server-to-server connectivity, the Fibre Channel interconnect protocol, an industry-networking standard, was developed in the early 1990s[KR00]. Fibre Channel is a high-speed (100Megabytes / Second) channel that is capable of large connection

distances measured in kilometers. The Fibre Channel protocol uses addressable nodes such that storage devices can be configured in a network fabric rather than point-to-point. Using compatible Fibre Channel Host Based Adapters (HBAs) any workstation can address the storage and access data to and from it just as it would do to a directly connected SCSI storage device. The network, which includes HBAs, hubs, switches and network-attached storage, is called a Storage Area Network (SAN). The distinct difference between using the conventional client/server distributed systems and a SAN is that any workstation connected to the network fabric can directly access the network-attached storage devices.

As pointed out by [KB99], existing local file systems such as Microsoft's NTFS, SGI's XFS and Apple's HFS assume that any visible storage, whether in a network or locally connected, is owned by the local workstation. If the file system accesses the storage without coordination with other workstations in the network and two or more inadvertently share access to the network-attached storage devices, data corruption can occur. To avoid data corruption and due to no readily available software solution, the early SAN devices were used only in a server-attached shared storage environment in which disk storage systems were connected to network file servers using the Fibre Channel interconnect protocol, but in order to access the storage devices client workstations still sent messages to servers via a local area network. These network architectures provided improved disk and interconnect performance but failed to solve many of the shortcomings of conventional client/server distributed computing models. They relied on network devices that could not maximize the data access performance

potential of high-speed interconnect technology enabled by Fibre Channel. Additionally, without SAN-enabled software, distributed computing cannot offer truly shared information access, as individual servers on the network still "own" disk space (islands of data) that cannot be directly accessed by other servers or users on the network. In the last three or four years however, a number of mostly third-party vendors have begun developing and offering distributed file system solutions that exploit the SAN distributed environment [O98], which are referred to SAN File Systems (SAN FS).

A SAN File System is architecture for distributed file systems based on shared storage that fully exploits the special characteristics of Fibre Channel-based LANs. The key feature of the SAN File System is that clients transfer data directly from the device across the SAN. Since a SAN File System allows more than one client to access the data from the same storage device, it must recognize the existence of other clients accessing the same storage device and file system data and meta-data. There are two prevailing approaches. One approach is to use a distributed lock manager and essentially treat the SAN nodes as a tight cluster, similar to a VAX Cluster. Global File System (GFS) [O98][SRO96] is a popular choice in this category, due to the fact that it is freely available on Linux. An alternative is a split data-metadata architecture. This architecture uses a single server or multiple servers as the coordinator of all the metadata traffic. File system metadata contains information such as a file's name, allocation information, security and ownership specifics, and other information about the file. This information is typically very small and easily communicated over a LAN

between peers. The file data, including the actual contents, is assessed directly from each SAN node, including the clients. This approach provides a high-performance, scalable solution for data sharing. Data access now happens at channel speeds rather than network speeds. Since the server is freed from the I/O traffic, it can handle many more clients than before, and is no longer a bottleneck in the data stream. The solution has potential to scale to the limits of the storage area network fabric.

SAN File Systems are currently available in products from EMC (MPFS), ADIC (CVFS)[KL99], DataDirect Networks (CDNA), SGI (CXFS)[KK99], and Tivoli Systems (SANergy)[SA00]. Furthermore, the Storage Networking Industry Association (SNIA) [WT00] File System Working Group has developed a similar architecture. Although this architecture has some drawbacks for very small files (because the metadata still involves the LAN), and it requires added coordination of the metadata, it is an efficient solution for large file transfers

EMC's Multiplex File System (MPFS) is a unique solution that provides high-bandwidth access to shared files from heterogeneous hosts directly attached to the storage system. The hosts interact with an MPFS server for synchronization, access control, and metadata management, but do data transfers directly to and from the storage device. Rather than replace traditional file access protocols, MPFS interoperates with, and uses, standard protocols such as NFS and CIFS for control and metadata operations. It adds a thin, lightweight protocol called the File Mapping Protocol (FMP) for exchanging file layout information between the hosts and the

MPFS server, thus allowing the hosts to directly read and write file data. Finally, by working with the standard file access protocols, MPFS offers robust, fully functional file sharing.

Achieving superior performance to NFS in terms of scalability and I/O throughput is the major design goal of MPFS. The MPFS split data-metadata architecture demands new means to evaluate performance. Current file system and I/O benchmarks are not suitable for MPFS performance evaluation. First of all, most current file system benchmarks only measure server performance. MPFS allows clients to directly access the storage subsystem in contrast to NFS or CIFS, in which the server is the only gateway to access the storage subsystem. Therefore, in order to measure MPFS performance, the benchmark should target both server and client. Second, most current file system benchmarks are protocol specific. They can be classified as either an NFS benchmark, such as SPEC SFS [SPEC97], or a CIFS benchmark, such as NetBench [ZD00]. Since MPFS is quite a different file access protocol, we can use neither of NFS or CIFS benchmarks to measure MPFS performance. Third, most current file system benchmarks cannot support multiple file access protocols. MPFS uses NFS or CIFS as underneath file access protocol for metadata operations. In order to measure MPFS performance and compare MPFS with other file access protocols, the benchmark should support multiple file access protocols. Fourth, most current file system benchmarks are not portable across different operating systems. MPFS supports various operating systems, such as Solaris, Windows NT, Linux, AIX, etc., so the ideal benchmark should be portable across various operating systems. In summary, since

MPFS is a totally new file access protocol, no current file system benchmarks can be used directly to measure MPFS performance, which motivates us to develop a new file system and I/O benchmark for MPFS performance evaluation.

Before we propose our own benchmark, we develop criteria for ideal benchmarks to evaluate MPFS performance.

First, an ideal benchmark should help in understanding system performance. The MPFS benchmark should help the system architects and operating system programmers to evaluate design changes and isolate reasons for poor performance. Standard benchmarks can only present the average system behavior. It is difficult for designers to use standard benchmarks to evaluate the isolate component's performance. Micro-benchmarks are easily correlated with implementation features, but they are not so good for predicting the performance of an entire system in actual use. The ideal MPFS benchmark should provide insight into MPFS performance not only in the system level but also in the isolated component level. Furthermore, since MPFS is a networked file system, the ideal MPFS benchmark should focus on the file system rather than the underlying storage subsystem. In other words, we cannot only measure the storage system's throughput and ignore other critical file system features, such as file caching, data sharing, etc.

Second, the ideal benchmark should be scalable and target both large and small files. As mentioned above, the major benefit of using MPFS is in improving scalability. The

ideal MPFS benchmark should allow multiple I/Os to be in progress simultaneously. Most current file system benchmarks do not scale with the number of processes issuing I/O and number of hosts that have concurrent I/O processes. In order to measure the file access for both small and large files, the file set for the ideal MPFS benchmark should be scalable, especially since the MPFS architecture may have some drawbacks for very small files (because the metadata still involves the LAN).

Third, the ideal benchmark should be relevant to a wide range of applications. MPFS can support a wide range of applications, such as Computer Aided Design (CAD), Medical Diagnostic Imaging, Streaming Video Web Servers, and Oil-Gas geological data analysis. To mimic those applications, the benchmark should dynamically adjust the workload parameter according to different performance characteristics of the applications. The easiest way to characterize an application is by using trace data. By characterizing the trace, we can get a set of workload parameters and using these workload parameters, we can generate a stochastic workload that can be executed as a benchmark. Characterizing the trace allows us to scale workloads and alter their composition yielding benchmarks that can help predict the system performance. The problem of using trace data is that little or no trace data is available that accurately represents current or future workloads. The ideal MPFS benchmark should allow the users to collect their own trace data for the specific applications and the benchmark should dynamically adjust the workload parameters accordingly.

Fourth, the ideal benchmark should provide workloads across various platforms. MPFS can support various operating systems, such as Solaris, Windows NT, AIX. The ideal MPFS benchmarks should be portable for different platforms.

Fifth, the ideal benchmark should provide data analysis rather than just collecting data. Besides providing performance data, the ideal MPFS benchmark should also help the designers to identify the performance problems. The benchmark should provide system monitors that collect performance statistics, analyze the data, and display the results. A well-designed monitor can help designers locate performance problems quickly and efficiently. An ideal MPFS benchmark system should also provide a database to record all performance data, which helps the designers to keep track of the system performance changes.

Sixth, the ideal benchmark should allow for fair comparisons across products. Different products have different implementations even though they use similar architectures. The workloads the benchmark generates should be general enough to make fair comparisons across products.

In this thesis, we propose a new benchmark for MPFS performance evaluation. Our benchmark consists of a number of application groups that can generate I/O intensive workloads. The application groups target some typical real-world MPFS applications and mimic them at the I/O level using a set of system calls in various combinations. The performance metrics of our benchmark includes some critical performance aspects

of MPFS: throughput, response time, scalability, file sharing, metadata operations and data transfer. Our benchmark is adjustable in terms of the application mix and can dynamically adjust the mix of I/O operations according to different application performance characteristics. Our benchmark is highly scalable in terms of file set selection. It can scale the file set to mimic some specific MPFS applications whose file transfers are normally huge, and for Internet applications that have many small files, our file set can be adjusted accordingly. To measure some critical aspects of MPFS performance, such as read and write, our benchmark has embedded micro-benchmarks that allow users to select certain isolated components to measure. Since our benchmark is an application level benchmark, it supports both UNIX clients and Windows NT clients. Like SPEC SFS 2.0 [SPEC97], our benchmark reports the average response time at the actual load that was delivered. We also report the throughput for some read or write operations that are specified by users. Our benchmark has a Web-based user interface that allows multiple users to access our benchmark system in order to specify the test parameters such as the application mix and file set size, run the benchmark and view the performance data. It also provides a database to save all the performance data.

In this thesis, we also measure the MPFS performance using our benchmark system and collect the performance data. Based on the performance data we collected, we use our benchmark to analyze the bottlenecks of MPFS performance.

Chapter 2 discusses the current file system and I/O benchmarks and shows how current file system and I/O benchmarks fall short of criteria we develop for the ideal MPFS

benchmark. Chapter 3 describes our benchmarking methodology in detail. Chapter 4 provides the system structure of our benchmark. Chapter 5 presents the performance data collected from our benchmark and discusses the MPFS performance bottlenecks based on those performance data. Chapter 6 summarizes our conclusions and presents possible future work.

Chapter 2 Related Work

In this section, we discuss the current file system and I/O benchmarks and show how these benchmarks fall short of the criteria we develop for the ideal MPFS benchmark.

There are two ways to categorize file system benchmarks. Based upon the methodology to construct benchmarks, benchmarks can be categorized into two classes: application benchmarks and synthetic benchmarks. Application benchmarks use standard programs, such as compilers, utilities, editors and databases, in various combinations, to produce a workload. Each benchmark targets a single application area, such as transaction processing or system development. Application benchmarks usually do a good job of accurately representing their target application area. The major drawback of application benchmarks is lack of generality or representativity. In other words, the application benchmark may work well for a particular class of applications but may not be suitable for other applications.

Synthetic benchmarks exercise an I/O system by directly issuing read and write commands. In contrast, application benchmarks use standard programs, which in turn issue I/O. By issuing reads and writes directly, synthetic benchmarks are able to generate more I/O intensive workloads. But, synthetic benchmarks often yield less convincing results because they, unlike application benchmarks, do not perform useful work.

Another way to categorize benchmarks is based on whether benchmarks measure overall system performance or isolated components' performance. Micro-benchmarks are constructed to measure the isolated components' performance in contrast to standard benchmarks that evaluate overall system performance. Micro-benchmarks are easy to use and can easily be correlated to specific implementation issues. Standard benchmarks to a certain extent reflect the overall system performance. However, these results do little to indicate how well a particular system will handle a particular application.

In the following, we briefly describe eight common benchmarks used in file system evaluation: Andrew[HKMN+88], TPC-B[TPCB90], SPECSFS2.0[SPEC97], NetBench[ZD00], self-scaling I/O benchmarks[CP93a][CP93b], IOStone[PB90], Bonnie[B90] and Application-Specific benchmark[SKS99]. We also show how these benchmarks fall short of the criteria we described in Chapter one for the ideal MPFS benchmark.

Andrew

Andrew was originally meant to be only a convenient yardstick for measuring file systems, not necessarily as a representative workload for benchmarking. It copies a file directory hierarchy, examines and reads the new copy, then compiles the copy. Since Andrew only gives a single bottom-line performance result, it cannot help us to understand the overall system performance.

TPC-B

TPC-B measures transaction-processing performance for a simple database update. TPC-B repeatedly performs *Debit-Credit* transactions, each of which simulates a typical bank account change on a bank database. TPC-B's main metric is maximum throughput measured in transactions-per-second.

As a transaction processing benchmark, TPC-B focuses on measuring the machine supporting database and database software. Even though TPC-B has an extremely well defined scaling strategy, it is not suitable for MPFS performance evaluation. The MPFS benchmark should focus on file system and I/O.

SPEC SFS 2.0

SPEC SFS 2.0 is a synthetic benchmark that uses a mix of NFS operations to stress a network server. SPEC SFS 2.0 is highly parameterized. Besides the percentage of each operation in the workload, SPEC SFS 2.0 gives a user the ability to change the number of clients issuing requests to the server, the rate at which each client issues requests, the total size of all files, the block size of I/O requests, and the percentage of write requests that append to an exiting file. The metric for SPEC SFS 2.0 is a throughput (NFS operations per second) versus response time graph.

SPEC SFS 2.0 measures NFS server performance by generating NFS client RPCs. It therefore cannot be used to compare NFS against other protocols, nor can it be used to test client-side caching strategies. Since MPFS is a totally different file access

protocol, we cannot use SPEC SFS 2.0 to measure MPFS performance. The MPFS benchmark we develop uses similar workload management scheme as that of SPEC SFS 2.0 but is quite different in the following aspects:

- SPEC SFS 2.0 measures NFS server performance; our benchmark measures MPFS system (include clients, server and storage subsystem) performance.
- SPEC SFS 2.0 uses RPC to generate an NFS workload; our benchmark uses application level operation groups to stress the server.
- SPEC SFS 2.0 does not share files and directories among multiple processes; our benchmark has dedicated operation group to measure file-sharing performance.
- SPEC SFS 2.0 has a fixed operation mix; our benchmark can adjust the mix of I/O operations according to different application performance characteristics.
- SPEC SFS 2.0 has the same file set and file access pattern for each NFS operation; our benchmark can adjust the file set and file access for different application groups.
- SPEC SFS 2.0 can only used in the UNIX platform; our benchmark is platform independent since it is application level benchmark. Our benchmark supports both Unix and Windows NT systems.
- SPEC SFS 2.0 randomly selects an operation; our benchmark is context sensitive in operation selection.
- SPEC SFS 2.0 only measures the overall system performance; our benchmark has embedded micro-benchmarks to measure specific MPFS performance aspects, such as sequential read operations.

- SPEC SFS 2.0 does not provide a well-designed user interface; our benchmark has a Web-based user interface that allows multiple users to access our benchmark system. Our benchmark system also provides a database to save all the performance data.

NetBench

NetBench V 6.0 is a portable I/O benchmark program that lets you measure the performance of file servers as they handle network file requests from clients. NetBench accepts as clients PCs running Windows 95/98 and Windows NT. NetBench takes leading applications for Windows-based PCs and profiles them to determine what sort of file requests they perform and how frequently they perform them. After profiling the applications, NetBench creates scripts to mimic the network file operations of those applications. NetBench works by performing a variety of network file operations across a large set of files in numerous directories. Its test files include data from text and binary files. It creates and deletes files, moves data in different-sized chunks, intersperses write and read operations, and works with multiple files at the same time. NetBench provides overall I/O throughput score and average response time for server and individual scores for the clients. NetBench reports its throughput results as megabits per second. It reports its response time results as milliseconds.

NetBench is an application level I/O benchmark that measures file server performance but only accepts as clients PCs running Windows 95/98 and Windows NT. It therefore

cannot be used to measure the system with clients running on UNIX platform. Our benchmark should support both UNIX and Windows platforms.

Self-Scaling Benchmark

Self-Scaling benchmark automatically scales its workload depending on the performance of the system being measured. During evaluation, the benchmark automatically explores the workload space, searching for a relevant workload on which to base performance graphs. To directly compare performance results for multiple systems, Self-Scaling Benchmark use a technique called **predicted performance** to estimate performance for unmeasured workloads.

Self-Scaling Benchmark is an I/O limited benchmark that is not suitable for measuring MPFS performance since we need to measure performance of both the file system and the I/O subsystem. Also, this benchmark is difficult to implement due to its complexity.

IOStone

IOStone is a synthetic I/O benchmark based on system traces of Unix minicomputers and workstations and IBM mainframes. One process performs all the accesses – no I/O parallelism is present. IOStone reports a single throughput result.

IOStone claims to be an I/O benchmark, but actually measures the memory subsystem; all of the tests fit easily in the cache since it only accesses 1 MB file set. The MPFS

benchmark should be I/O intensive and access much larger file set than IOStone does. Like Andrew, IOStone only gives a single bottom-line performance result.

Bonnie

Bonnie runs six different workloads that show the performance difference between reads versus writes and block versus character I/O. One workload sequentially reads the entire file a character a time; another writes the file a character at a time. Other workloads exercise block-sized sequential reads, writes, or reads followed by writes (rewrite). The final workload uses three processes to simultaneously issue random I/Os. For each workload, Bonnie reports throughput, measured in KB per second or I/Os per second.

Therefore, Bonnie is actually a disk benchmark. Bonnie focuses on disk I/O but does not vary other aspects of the workload, such as the number of concurrent I/Os. The MPFS benchmark should be a benchmark that measures both disk subsystem and file system. The MPFS benchmark should also be able to scale the number of processes issuing I/O and number of hosts that have concurrent I/O processes.

Application-Specific benchmark

The goal of the Application-Specific benchmark is to evaluate the performance of complex systems in the context of specific applications. The benchmark uses three different approaches to application-specific measurement, one using vectors that

characterize both the underlying system and an application, one using trace-driven techniques, and a hybrid approach. Since the Application-Specific benchmark works well for certain specific applications the benchmark target but not other applications. The MPFS benchmark should target a variety of applications.

In summary, current I/O and file system benchmark suites fall short of the criteria we apply to the ideal MPFS benchmark. This suggests the need for a new benchmark to measure the MPFS performance.

Chapter 3 Approach

This chapter describes the approaches to construct our MPFS benchmark in detail. Section 3.1 provides an overview of our approach. Section 3.2 introduces the performance metrics for MPFS. Section 3.3 describes our load generation mechanism.

3.1 Overview

In order to develop an MPFS benchmark, we first define the MPFS performance metrics. Based on the performance metrics we define, we construct the application groups that measure the critical performance characteristics of MPFS. The application groups target both I/O operations and metadata operations for MPFS. We derive the application mix percentage from the low-level NFS operation mix percentage since MPFS uses the NFS protocol underneath for control and metadata operations. File set construction is another critical step to developing our MPFS benchmark. The file set for our MPFS benchmark is application-based, namely each application group has its own file set. The file access pattern for our MPFS benchmark is based on an earlier file access trace study [RLA00]. There are five major issues we consider when we design the file access patterns for our benchmark: file access order, file access locality, file access burst, read/write ratio and overwrite/append ratio. Our benchmark uses a different load management scheme from that of LADDIS[WK93], in which the think time between requests is exponentially distributed and the load-generating processes in each load-generator is Poisson distributed. The major purpose of using this scheme is to mimic a real-life workload. Our benchmark chooses the next operation to perform

based on a probability derived from the operation mix and an operation context map that follows the common operation sequence. In order to measure the data sharing performance for MPFS, our benchmark has five application groups that are dedicated for this purpose. To measure how MPFS performs under I/O intensive traffic, our benchmark has embedded micro benchmarks, which include sequential read, sequential write, random read, random write and random read/write.

3.2 Performance Metrics for MPFS

Before we develop a benchmark for MPFS performance evaluation, we need to define the metrics for MPFS performance. We list 3 metrics that are critical to MPFS performance evaluation.

Throughput

Throughput is a measure of the rate at which the storage system delivers data. Throughput of a storage device is measured in two ways: I/O rate, measured in *operations/second*, and data rate, measured in *bytes/second* or *megabytes/second*. I/O rate is generally used for applications where the size of each request is small, such as transaction processing [CP93a]; data rate is generally used for applications where the size of each request is large, such as scientific applications [CP93a]. Since the request size for MPFS applications may vary tremendously, we use both I/O rate and data rate to measure the throughput.

Response Time

Response time is a measure of how long a storage system takes to access data. The performance of many modern applications depends on the speed at which the system can respond to an asynchronous stream of independent and diverse events that result from interactive user input or network packet arrival. In our benchmark, we measure the overall average response time for all mixed operations and the average response time for individual operation, both of which are measured in *milliseconds per operation* (Msecs/Op).

Scalability

Scalability is a measure of how well a system responds as the number of clients increases. Improved scalability is one of the key design goals of MPFS. The central idea behind the MPFS architecture is split data and metadata. Clients can collect to storage devices directly and leave servers manage metadata traffic, which allows data access at channel speeds rather than at network speeds. Since the server is freed from the I/O traffic, it has the potential to handle many more clients than traditional networked file systems, and is no longer a bottleneck in the data stream. MPFS should be able to scale to the limits of the storage area network fabric. There are various scalability metrics. In our benchmark, we use *number of client hosts* supported by the system with acceptable performance as the scalability metric.

3.3 Workload Generation

3.3.1 Application Group Construction

Our application groups allow us to predict the critical performance characteristics of MPFS, especially for read/write operations and scalability. We create a total of 13 application groups where each group consists of a sequence of file I/Os or meta-operations. Since MPFS has separate metadata and real data paths, our application groups include both I/O operation groups and metadata operation groups. Each application group is independent of each other. Our benchmark uses a sequence of application groups with various combinations to generate workloads.

3.3.1.1 I/O Operation Groups

The I/O operation groups focuses on file I/O operations. Since data access now can happen at channel speeds rather than at network speeds, MPFS should have significant improvement in the I/O throughput and latency. The purpose of performance measurement in this aspect is to test how the throughput and response time of I/O operations vary under different circumstances, such as different operation types and modes, different volume management mechanism, different file sets, different caching policies, etc.

We have total of seven I/O operation groups. They include *read, write, read-write, memory mapping, read sharing, write sharing and read-write sharing.* The read application group consists of read, fread and direct read. The write application group consists of write, truncate, fwrite, direct write and data flush. Each write performs two

distinct types of operations: writes that overwrite existing data in a file, and writes that append new data to the end of a file. The read-write application group is a mix of read and writes according to a pre-defined read to write ratio. The memory-mapping group consists of mmap and munmap. In the recent years, memory mapping has become a common method to access files, especially for shared libraries. The read and write sharing groups will be described in section 3.3.6.

3.3.1.2 Meta Operation Groups

The metadata operation groups focus on file metadata operations. Since MPFS has a separate data and metadata path, the performance of metadata operations in terms of latency and throughput is an important metric to evaluate MPFS performance. The major issue is that the high overhead of metadata operations may greatly degrade the system performance.

We have total of 6 metadata operation groups. They include *file stat, get attribute, set attribute, directory,* and *symbolic link*. The file stat group includes stat, fsstat, statvfs and fstatvfs. The get attribute group contains getacl, fgetacl, access and getaccess. The set attribute group includes chdir, mkdir, rmdir and readdir. The symbolic link group includes symlink, readlink, link and unlink.

3.3.1.3 Windows NT Operation Groups

The operation groups for Windows NT contain 16 I/O operations, which follow the file I/O calls used in the Disk Mix test of NetBench [ZD00]. The 16 I/O operations are

determined by profiling 9 leading applications for Windows-based PCs [ZD00]. The 16 I/O operation groups are *Read, Write, Open File, Close, Lock, Unlock, Delete File, Create New File, Set File Attributes, Get File Attributes, Find First, Find Next, Rename File, Get File Time, Flush File Buffers*, and *Get Disk Free Space*.

3.3.2 Application Mix Tuning

We have a total of 13 application groups that consist of 24 independent file operations, among which there are 16 meta-operations and 8 I/O operations as described in 3.3.1. The mix of these 24 file operations generates the workload to stress the MPFS system. The mix percentage of these 24 file operations is derived from the low-level NFS or CIFS operation mix percentage since MPFS uses the NFS or CIFS protocol underneath for control and metadata operations.

3.3.2.1 Application Mix Tuning based on NFS Operation Mix

The NFS operation mix percentage we use is the NFS version 3 mix published by SPEC SFS 2.0 [SPEC97].

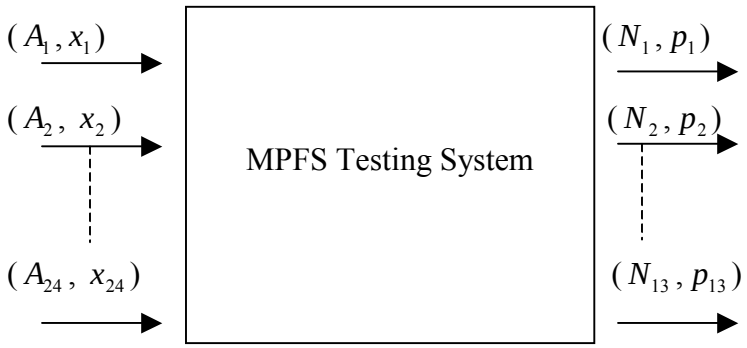

Figure 3.1 Derivation of the application mix from the NFS operation mix

Figure 3.1 shows how we can derive the application mix from NFS operation mix. $(A_n,\ x_n)$ represents the file operation A_n, where $n = 1, 2, \Lambda\ , 24$ corresponding to the 24 file operations as described in 3.3.1, and the corresponding mix percentage x_n, where $x_1 + x_2 + \Lambda + x_{24} = 1$. $(N_n,\ p_n)$ represents the NFS operation N_n, where $n = 1, 2, \Lambda\ , 13$ corresponding to 13 NFS operations defined by NFS Version 3, and the corresponding mix percentage p_n, which are obtained in the server side using the NFS profiling utility such as nfs_stat. First, we employ one application group, say A_1, to the MPFS Benchmark. The NFS profiling utility in the server side captures the network trace with which we can calculate the corresponding NFS operation mix percentage:

$$(p_{1,1},\ p_{1,2}, \mathrm{K}\ , p_{1,13})$$

Continuing the same procedure for other application groups, we can obtain all NFS operation mix percentages that form the following array:

$$P = \begin{bmatrix} P_{1,1} & P_{1,2} & \Lambda & P_{1,12} & P_{1,13} \\ P_{2,1} & P_{2,2} & \Lambda & P_{2,12} & P_{2,13} \\ \mathrm{M} & \mathrm{M} & \mathrm{O} & \mathrm{M} & \mathrm{M} \\ P_{23,1} & P_{23,2} & \Lambda & P_{23,12} & P_{23,13} \\ P_{24,1} & P_{24,2} & \Lambda & P_{24,12} & P_{24,13} \end{bmatrix} \qquad (3.1)$$

Second, assuming MPFS testing system is a linear system, we have the following linear equation:

$$\begin{bmatrix} P_{1,1} & P_{1,2} & \Lambda & P_{1,23} & P_{1,24} \\ P_{2,1} & P_{2,2} & \Lambda & P_{2,23} & P_{2,24} \\ \mathrm{M} & \mathrm{M} & \mathrm{O} & \mathrm{M} & \mathrm{M} \\ P_{12,1} & P_{12,2} & \Lambda & P_{12,23} & P_{12,24} \\ P_{13,1} & P_{13,2} & \Lambda & P_{13,23} & P_{13,24} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \mathrm{M} \\ x_{19} \\ x_{24} \end{bmatrix} = \begin{bmatrix} p_1 \\ p_2 \\ \mathrm{M} \\ p_{12} \\ p_{13} \end{bmatrix} \qquad (3.2)$$

Solving this equation, with the constraint:

$$x_1 + x_2 + \Lambda + x_{24} = 1 \qquad (3.3)$$

We can obtain the application mix percentage.

It is very hard to characterize this linear equation and present the solution using the conventional method. Instead we develop an algorithm to tune the application mix percentage to approximate to the desired one. Our algorithm tries to solve the following problems:

Given the desired NFS operation mix percentage, which is either the NFS version 3 mix published by SPEC SFS 2.0 or the one specified by the users, find the file operation mix percentage $(x_1, x_2, \Lambda\ , x_{24})$ *such that the underlying NFS mix percentage approximates the desired NFS operation mix percentage.*

Our algorithm is based on the observation that a solution to the linear equation (3.2) shall lies in the compact region R defined by:

$$R = \left\{(x_1, x_2, \Lambda\ , x_{24}) : 0 \le x_i, \sum_{i=1}^{24} x_i = 1, i = 1, 2, \Lambda\ , 24\right\} \qquad (3.4)$$

For each point $(x_1, x_2, \Lambda\ , x_{24})$ in the region R, there is an underlying NFS operation mix percentage $P = (p_1, p_2, \Lambda\ p_{13})$, which shall satisfy:

$$\min_{p \in R} D(P, \varphi) = 0 \qquad (3.5)$$

Where φ is the desired NFS operation mix percentage. D is the Euclidean distance between P and φ defined by:

$$D(P,\varphi)=\sqrt{\sum_{j=1}^{13}\left(p_j-\varphi_j\right)^2} \qquad (3.6)$$

The idea is that the algorithm should search for the root of the linear equation (3.2) by computing $D(P,\varphi)$ for all points $(x_1, x_2, \Lambda\ , x_{24})$ in R. This algorithm does not require an initial guess and it gives an answer that will satisfy the inequality constraints in (3.4).

The algorithm attempts to find a point in R that achieves the minimum in (3.5). However, the computer cannot scan the bounded region R continuously but must do so in increments of size $\delta > 0$ for each x_i coordinate for $i = 1,\Lambda\ ,24$. Therefore, in practice, we compute $D(P,\varphi)$ only for points $(x_1, x_2, \Lambda\ , x_{24})$ belonging to a finite lattice of isolated points in R, and not for all points in R. After each increment is made and the new $D(P,\varphi)$ is computed, the running minimum value and its coordinate are updated. Once the lattice is exhausted, the final minimum value and its coordinates are declared as the final answer. Thus, in the vast majority of the cases the algorithm can only give an approximation to the root(s) of linear equation (3.2). If in the course of computing $D(P,\varphi)$ and updating the running minimum, $D(P,\varphi)$ becomes negligible, i.e., $D(P,\varphi) \le \varepsilon$, the program stops and declares $(x_1, x_2, \Lambda\ , x_{24})$ as the solution.

The algorithm is described as follows:

1. Initialize $[x_1, x_2, \Lambda\ , x_{24}]$

In this initialization stage, each mix percentage $x_i, 1 \le i \le 24,$ is assigned a starting number, which should meet the constraint:

$$x_1 + x_2 + \Lambda + x_{24} = 1$$

2. Tuning $[x_1, x_2, \Lambda, x_{24}]$

In this tuning stage, we adjust $[x_1, x_2, \Lambda, x_{24}]$ to let the distance between the underneath NFS operation mix percentage and the desired NFS operation mix percentage under a predefined threshold γ.

The following is the pseudocode of the computer program implementing the tuning procedure. In this program, the user must specify the step size δ and the threshold γ. We use P^k to represent the underlying NFS operation mix percentage in the k th iteration that is captured using the NFS profiling utility NFS_Stat and $D(P^k, \varphi)$ the Euclidean distance between P^k and the desired NFS operation mix percentage φ.

***k* ←0**

***for i* ← 1 to 23**

***for j* ← i+1 to 24**

while $D(P^k, \varphi) \le D(P^{k-1}, \varphi)$ ***and*** $x_j \ge 0$

do $x_i \leftarrow x_i + \delta$ ***and*** $x_j \leftarrow x_j - \delta$

***k* ← k+1**

if ($D(P^k, \varphi) \le \gamma$ ***)***

then tuning procedure terminates

if $D(P^k, \varphi) > D(P^{k-1}, \varphi)$

while $D(P^k, \varphi) \le D(P^{k-1}, \varphi)$ ***and*** $x_i \ge 0$

do $x_i \leftarrow x_i$ ***-δ and*** $x_j \leftarrow x_j + \delta$

$k \leftarrow k+1$

if ($D(P^k, \varphi) \le \gamma$ ***)***

then tuning procedure terminates

Figure 3.2 The pseudocode of the tuning procedure

The tuning procedure tunes the mix percentages in pairs. In the k th iteration, the profiling tool captures the underlying NFS operation mix percentage P^k. If $D(P^k, P) \le \gamma$, namely the Euclidean distance between P^i and the desired NFS operation mix percentage is within the predefined threshold, the tuning procedure terminates. Otherwise, x_i is incremented by the step size δ. Meanwhile, to meet the constraint: $x_1 + x_2 + \Lambda + x_{24} = 1$, x_j is decremented by the same step size. The comparison between the current NFS operation mix percentage P^k and previous one (P^{k-1}) can determine the further tuning direction. If $D(P^k, \varphi) < D(P^{k-1}, \varphi)$, which means the operation mix percentage is approaching the target mix percentage, the tuning procedure continues the same direction. Otherwise, x_i and x_j are tuned in the opposite direction, namely x_i is decremented by the step size δ and x_j is incremented by the step size δ. This tuning procedure continues untill a desired mix percentage is found or all the possible tunings have been exhausted. Obviously, the approximations

improve monotonically as the step size δ gets smaller and smaller. However, the smaller the step size, the longer the tuning process takes. Hence, there is a trade-off between the speed computation and the accuracy of the results.

3.3.2.2 Application Mix Tuning based on CIFS Operation Mix

As mentioned in section 3.3.1.3, the operation groups for Windows NT contain 16 I/O operations, which follow the file I/O operations used in the Disk Mix test of NetBench. NetBench took the leading applications for Windows-based PCs and profiled them to determine what sort of file requests they performed and how frequently they preformed them. Since NetBench is the leading benchmark for CIFS and MPFS for Windows NT uses CIFS as a protocol underneath, our benchmark adjusts the application mix percentage to match the underneath CIFS operation percentage, which is obtained by profiling NetBench. The reason we profile NetBench is that, unlike SPEC SFS, NetBench did not publish its CIFS operation mix percentage. We captured the CIFS operation mix percentage in NetBench by running NetBench against the EMC Celerra file server and using our own profiling tool to capture the underneath CIFS operations. Using the similar tuning procedure we described above, we tune the application mix percentage for the 16 operations.

3.3.2.1 Application Mix Tuning based on User Specification

One of our benchmark's major design goals is to provide the users a way to get a realistic measure of MPFS performance for their specific applications. Our benchmark allows the users to collect the trace data for their specific applications and profile them to capture the underneath NFS or CIFS operation mix percentage. Using the tuning

procedure we described above, our benchmark can adjust the application mix percentage to match the underneath NFS or CIFS operation mix percentage the users specify.

3.3.3 File Set Construction

3.3.3.1 Application-Based File Set

The file set for our benchmark is application-based, namely each application group has its own file set that is different from others. The reason that different application groups have different file sets lies on two folds. First of all, different application groups may have different requirements for their file set. For instance, the file set for read or write application groups are quite different from that for directory or symbolic link application groups. Secondly, according to [RLA00], files tend to have a bimodal access pattern; they are either read-mostly or write-mostly and very few files are both read and written, which means read and write groups should access different file sets.

3.3.3.2 File Set for read and write application groups

The file set for both read and write application group contains three subsets that have different file size distributions. The first file subset is called the small file set, which has the file size distribution published by SPEC SFS 2.0 [SPEC97] (see table 3.1).

File Size (bytes)	Percentage (%)
1k	33
2k	21
4k	13
8k	10
16k	8
32k	5
64k	4
128k	3
256k	2
1M	1

Table 3.1 The file size distribution for small file set

According to [RLA00], most accessed files are small, but the size of the largest files continues to increase. To reflect this trend and also consider the fact that a large number of MPFS applications access large files, we create the medium and large file sets. The file size distribution for both file sets are shown in Table 3.2 and 3.3.

File Size (bytes)	Percentage (%)
1k	20
4k	10
16k	36
64k	10
128k	10
256k	6
1M	4
8M	2
128M	2

Table 3.2 The file size distribution for the medium file set

File Size (bytes)	Percentage (%)
16k	10
128k	20
512k	20
1M	20
16M	10
128M	8
512M	6
1024M	3
2048M	3

Table 3.3 The file size distribution for the large file set

Read and write groups access separate file sets but have the same file size distributions.

3.3.3.3 File Set for metadata operation groups

For operations that create and remove files, our benchmark incorporates a number of simplifying assumptions; namely our benchmark does not initiate operations that are expected to fail. To avoid performing failed operations, our benchmark explicitly manages file name space for those application groups and maintains existence state information for each file.

Directory and symbolic link application groups are performed on 10 file sets, each of which has certain number of directories and symbolic links that are determined by the target load level as described in Section 3.3.3.4.

3.3.3.4 File Set Scaling

The number of files and amount of data in our file set is scaled to the target load level specified for the benchmark run at the rate of 10 megabytes (MB) of data for each Operations/Second (Ops/Sec) of load. For the directory application groups, the number of directories in the file set is also scaled to the target load level at the rate of 10 directories for each Operations/Second (Ops/Sec) of load. For the symbolic link application groups, the number of symbolic links in the file set is scaled to the target load level at the rate of 100 symbolic links for each Operations/Second (Ops/Sec) of load.

3.3.4 File Access Pattern

As mentioned in section 3.3.3.1, our file set is application-based; namely, each application group has dedicated file set to access. In this section, we discuss the file access pattern such as whether a file is read or written and the order in which its bytes are accessed. We address some critical issues regarding file access: file access locality, access distribution, and file access bursts.

3.3.4.1 Read and Write Patterns

There are five major issues we consider when we designed the read and write patterns for our benchmark: file access order, file access locality, file access burst and overwrite/append ratio. In the following, we discuss each pattern in details.

File Access Order

We define two types of file access orders, namely sequential access and random access. We classify a file access as sequential if it reads or writes a file in order from

beginning to end, and random otherwise. According to [RLA00], small files tend to be accessed sequentially, while large files tend to be accessed randomly, which we use as a guideline for file selection. In other words, we choose a small file for sequential access and a large file for random access.

File Access Locality

File access locality refers to the fact that the same files tend to get the same type of access repeatedly. According to [RLA00][RW93], overwrite and read have significant locality, namely the same files tend to get overwritten over and over again and many files are repeatedly read without being written. To emulate the overwrite locality, our benchmark chooses the most currently overwritten files for the next overwrite. The read locality is achieved by creating separate file sets for read and write operation groups and choosing the most currently read files for the next read operation.

File Access Burst

File access burst refers to the fact that certain file access pattern occurs in bursts. According to [RLA00], write operations tend to occur in bursts. To emulate the write burst, a portion of each write operation group comprises a sequence of consecutive write operations.

Overwrite/Append Ratio

Our benchmark performs two distinct types of write operations: writes that overwrite existing data in a file, and writes that append new data to the end of a file. These operations exercise different parts of the server's file system implementation. Depending on data buffer alignment, over-writing data may require pre-fetching data

from the old data before overwriting part of it, while appending new data may require allocating space from the file system's pool of free storage space. To create an I/O intensive workload to measure MPFS I/O performance, the default parameters of our benchmark specify that 20% of write operations over-write existing data and 80% of write operations append new data.

3.3.4.2 Meta Data Access Patterns

The file access pattern for metadata operation groups follows a non-uniform, file access distribution algorithm that is implemented in LADDIS [Wk93]. During our benchmark initialization phase, the file set is partitioned into small groups of files or directories, and each group is assigned a probability of providing the next file or directory to be accessed. The probabilities are based on a Poisson distribution. Employing a Poisson distribution for file access, according to [Wk93], can increase file access locality. Each group contains the same number of files or directories, and the files or directories within each group have an equal chance of being selected.

3.3.5 Work Load Management

Our benchmark uses very different load management scheme from that of LADDIS [WK93]. The LADDIS manager process distributes the load equally among the load-generating systems, which, in turn, distributes the load equally among that system's load-generating processes. In order for each load generating process to produce the workload at the specified load rate, the server's average response time is estimated periodically. Our benchmark uses a different scheme, in which the think time between requests is exponentially distributed and the load-generating processes in each load-

generator is Poisson distributed. The arrival rate is the load level measured in *ops/sec*. The major purpose of using Poisson distribution is to mimic the real-life workload.

3.3.5.1 Load Distribution

Our benchmark distributes the load equally among the load-generators as LADDIS does, but instead of creating a fixed number of processes that share the workload equally, our benchmark creates a pool of threads, each of which performs certain operation asynchronously. The main thread in each load-generator waits a period of time, which is computed from an exponential distribution, and selects an idle thread to perform the next operation. After a thread finishes its task, it becomes idle and is put back to the thread pool.

3.3.5.2 Think Time Calculation

The think time is computed from an exponential distribution. We use a random-variable generation algorithm to generate an exponential variable as follows:

$$t = -\frac{1}{\lambda}\ln(u)$$

Where u is a random variable uniformly distributed between 0 and 1 and λ is a load level measured in *ops/sec*, which is specified by the user.

3.3.5.3 Operation Selection

Our benchmark chooses the next operation to perform based on a probability derived from the operation mix and an operation context map that follows the common operation sequence.

We build a hash-table in which each entry records the number of times each operation has been performed and contains a linked list that forms the operation context for that operation. The operation context follows the common operation sequence, e.g., a READDIR followed by a GET ATTRIBUTE, or a LOOKUP followed by a READ.

Using the hash-table described above, our benchmark can determine whether a selected operation has been performed the target number of times and what operation should be selected next. If a selected operation has reached its target, our benchmark will randomly choose another operation to perform. If a selected operation has not reached its target yet, our benchmark will perform it and proceed to the next operation in its linked list.

3.3.6 Read and Write Sharing

Sharing measures the system throughput and response time when multiple clients access the same data. Since MPFS allows clients to access the shared data directly, they need to provide some mechanisms to prevent data corruption. The key issue here is how those mechanisms affect the MPFS file system performance in terms of throughput and latency. We list three sharing semantics that our benchmark needs to support:

- **File System Sharing**
  - Multiple processes on multiple hosts reading files in the same file system.
  - Multiple processes on multiple hosts reading and writing files in the same file system.

- **Directory Sharing**
  - Multiple processes on multiple hosts manipulating the same directory.
  - Multiple processes on single host manipulating the same directory**.**
- **File Sharing**
  - Reads and writes to a file, which is opened by one and only one, process.
  - Reads and writes to a file where all processes with that file open reside on the same host.
  - Multiple processes on multiple hosts reading the same file.
  - Multiple processes on multiple hosts reading and writing the same file

Our benchmark has four application groups associated with the performance measurement for data sharing: read sharing in a single client, read sharing in multiple clients, write sharing in a single client and write sharing in multiple clients. These four application groups can be used as embedded micro-benchmarks or mixed with other application groups. We have a dedicated file set for these four application groups.

3.3.6.1 Read or write Sharing in a Single Client

In these two application groups, multiple processes in a single client read or write the same file simultaneously. To synchronize all the processes, our benchmark selects a process to be primary process to coordinate the read or write sharing. When the primary process chooses one of these two application groups, it will send signals to other processes that are either performing a file operation or waiting for the next operation. For those processes that are performing a file operation, they will finish the

current file operation and then wait for a go-ahead signal from the primary process. For those processes that are waiting for the next operation, they will stay in the waiting stage untill they receive a go-ahead signal from primary process. When the primary process detects all the other processes are ready, it will send a go-ahead signal to all of them to start the read or write operations.

3.3.6.2 Read or Write Sharing in Multiple Clients

In these two application groups, multiple processes in different clients read or write the same file simultaneously. To synchronize all the processes from different clients, our benchmark selects a client to be primary client to coordinate the read or write sharing. If a process in the primary client chooses one of these two application groups, it will send a RPC request to other clients. Once a client receives the request, it will pick one process that is currently idle or the next available process. The selected process will stay in idle till it receives the go-ahead signal from the primary client. When the primary client detects all the clients are ready, it will send a go-ahead signal to all of them to start the read or write operations.

3.3.7 Embedded Micro-Benchmarks

To measure how MPFS performs under intensive I/O traffics, our benchmark has embedded micro benchmarks including sequential read, sequential write, random read, random write and random read/write. For each type of testing, a client creates its own private file. It then reads from and writes to the file based on the test type and parameters you specify. Our benchmark reports the throughput measured in megabytes/second for each I/O test.

3.3.7.1 Sequential Read

1. During initialization, each client creates a set of test files of the specified file size and the data pattern specified in the directory indicated by the client path name. The number of test files is equal to the number of processes running in that client.
2. During the run phase, each process in the client seeks to offset 0 in its test data file.
3. Each process reads data from the beginning to the end.
4. When the process reaches the end of the file, it delays for the specified "think" time.
5. If iterations are remaining in the test, the client returns to step 1 and re-creates a set of test files of the size incremented by the specified step size.

3.3.7.2 Sequential Write

Sequential write follows the similar procedure as that of sequential read except for:

- The operation is writing a file.
- If appending is chosen as the type of write operation, the file size should be set to zero.

3.3.7.3 Random Read

Random read follows the similar procedure as that of sequential read except for:

- The offset is a random number between zero and the file size.
- The data size for reading varies with iteration.

3.3.7.4 Random Write

Random Write follows the similar procedure as that of sequential read except for:

- The offset is a random number between zero and the file size.
- The data size for writing varies with iteration.

3.3.7.5 Random Read/Write

For the random read/write, the process in the client performs both random read and random write operations based on a specified read/write ratio.

3.3.8 Caching

Since our benchmark measures the overall MPFS system performance including server and clients, the variations in client hardware and MPFS implementation effects such as caching may affect the accuracy. In order to avoid caching effects, our benchmark creates huge file set for each application group and makes the file selection as random as possible.

Each file set is partitioned into small group of files, and each group is assigned a probability of providing the next file to be assessed. The probabilities are based on a Poisson distribution. Each group contains the same number of files, and the files within each group have an equal chance of being selected.

To eliminate the caching effects due to the file set creations in each client, our benchmark use separate clients to create file sets. Using our caching avoidance schemes described above, the cache hit rates in both client and server sides are kept

below 10%. As mentioned in section 3.3.4.1, certain file operations have significant locality, such as overwrite and read. To maintain these file access localities, our caching avoidance schemes do not apply to overwrite and read operations.

Chapter 4 System Architecture

4.1 Overview

Based on the benchmark we described in chapter 3, we develop a ***MPFS Performance Monitor for EMC Celerra File Server (CFSPM)***. CFSPM is a benchmark program to measure the performance of Celerra File Server under MPFS protocol with a web-based interface. CFSPM can also be used as a testbed to evaluate our MPFS benchmark and to compare the MPFS protocol with the NFS and CIFS protocols. CFSPM provides the MPFS developers with testing tools to identify the MPFS performance bottlenecks. CPFSPM accepts clients running both Windows NT and Unix. To allow users to monitor the system activities while the performance test is running, CFSPM provides users with various monitors including a network monitor, I/O monitor, CPU monitor and memory monitor. These monitors can also help users to identify performance bottlenecks. To allow users track and compare the performance of a Celerra file server running different versions of MPFS, CFSPM uses a database to save the test results and testing configurations. Users can use this database to search for the performance data for a specific server or client package.

4.2 System Configuration

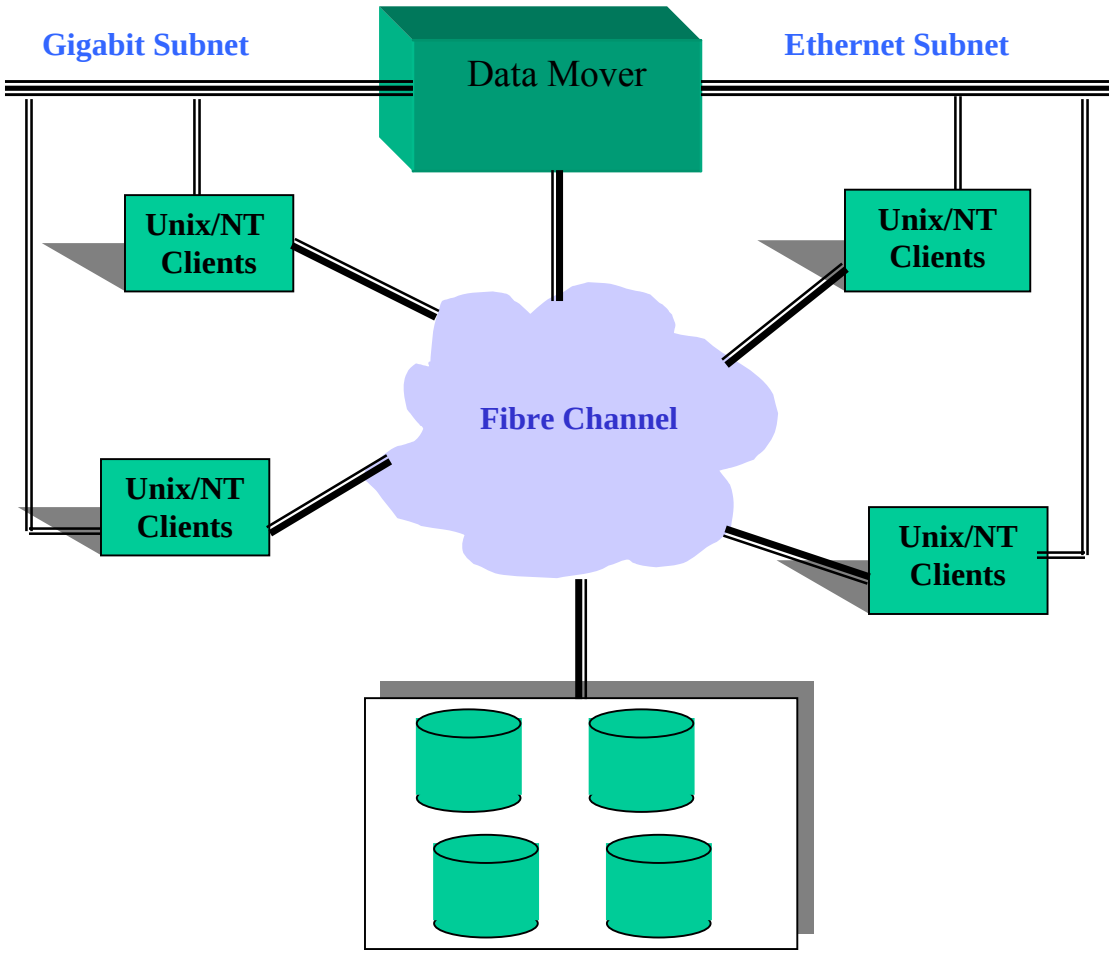

Figure 4.1 CFSPM System Architecture

Figure 4.1 shows the CFSPM system architecture. CFSPM consists of 40 Solaris clients and 40 Windows NT clients acting as load generators. These load generators are distributed over four LAN segments; two Gigabit Ethernets and two Ethernet 100BT. The Solaris clients and Windows NT clients access different file servers. There are two primary clients that coordinate execution of the load generation. The I/O subsystem of

CFSPM comprises two EMC Symmetrixs (storage device), each of which has direct connections to those load generators through a fibre channel network.

4.2.1 Server Configuration

CFSPM system currently uses two EMC Celerra Data Movers (***data mover 507***), which serve as file servers. Each data mover has 44 file systems, among which half are striping and the rest are non-striping. Each data mover has three network interfaces that connect to a Gigabit subnet, an Ethernet 100BT subnet and EMC backbone respectively. The connection between the data mover and Symmetrix is using the fibre channel.

4.2.2 Client Configuration

CFSPM system has two types of clients: PC clients running Windows NT or Windows 2000 and SUN Ultra-60 clients running Solaris 5.8. Each client is connected either to a Gigabit subnet or to an Ethernet 100BT subnet. The connection between client and symmetrix is using a fibre channel.

4.2.3 Network Configuration

CFSPM has four subnets: two Gigabits and two Ethernet 100BTs. One Gigabit subnet and one Ethernet 100BT subnet are dedicated to Solaris clients and the other two are dedicated to Windows clients. These subnets are physically isolated from each other and also isolated from other network systems.

4.2.4 Storage Subsystem Configuration

CFSPM has two Sym-5 Symmetrixs, which are EMC storage products. Each Symmetrix connects to clients and data movers through the fibre channel.

4.3 System Monitors

The system monitors are the critical parts for the CFSPM system, monitoring the testing progress and also collecting the statistics for both data mover and clients. Currently, the system monitors provides the following monitoring functionalities:

4.3.1 Network Monitor

The Network Monitor can capture the network traffic statistic data of a LAN segment. The network traffic statistic data includes input and output packet rate, packet error rate and collision rate. We define a set of rules to indicate the states of the network based on the packet collision rate, out-coming packet delayed and input and output errors per second.

4.3.2 I/O Monitor

The I/O monitor watches the disk I/O activities and collects the corresponding statistic data. The statistic data includes the data transfer rate for read and write operations and the disk utilization. We also define a set of rules to indicate the state of I/O in a client machine based on the disk utilizations and disk service time. If the disk is more than 5% but less than 20% busy and has service time of more than 30ms and less than 50 ms, the I/O monitor will send disk busy or overloaded warning. If the disk is more

than 20% busy and has service time of more than 50 ms, the I/O monitor will send the message to indicate that the disk may have problems.

4.3.3 CPU Monitor

The CPU Monitor watches the CPU usages. We define a set of rules to indicate the states of CPUs in a client machine based on its CPU usage. There are five CPU states we define:

4.3.4 Protocol Statistic Monitor

Protocol Statistics Monitor collects the statistics for all the file access protocols CFSPM supports including NFS, CIFS and MPFS.

4.4 Web-Based User Interface

CFSPM provides a Web interface to allow multiple users to specify the test parameters, run the test, monitor the test system, collect and save performance data. The CFSPM Web interface is implemented using CGI and Perl. The interface is shown below:

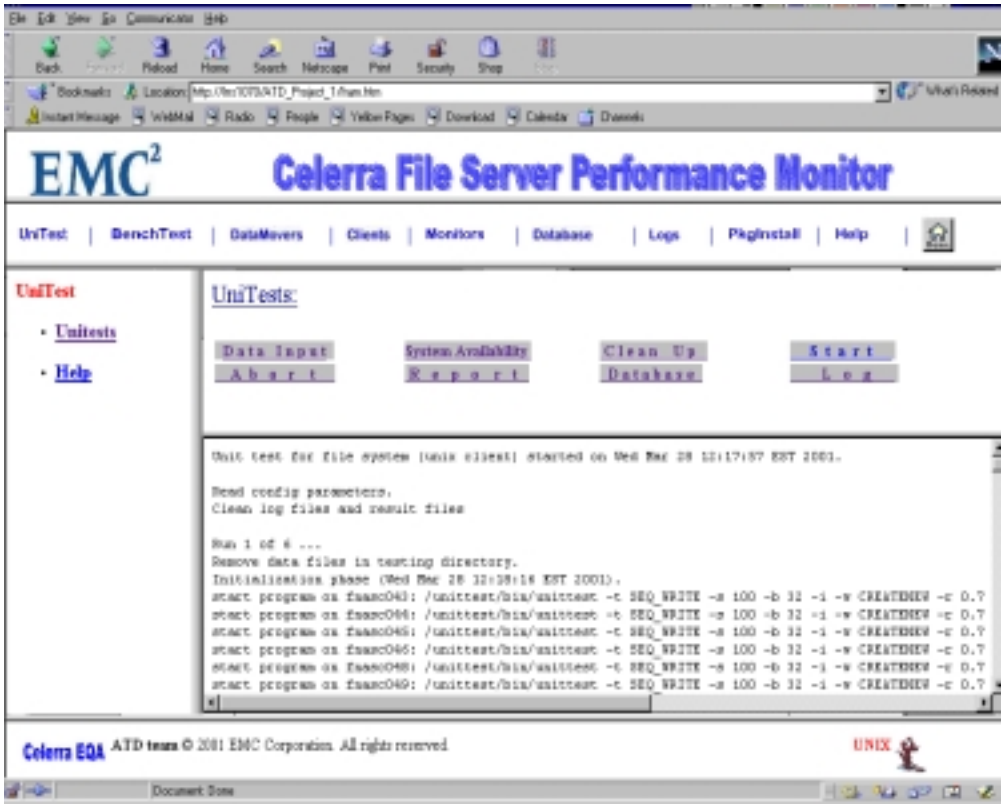

Figure 4.2 CFSPM Web Interface

Figure 4.2 shows the CFSPM Web interface. The functional items are briefly described below:

UniTest The embedded I/O micro-benchmarks.

BenchTest The MPFS benchmark.

DataMovers Utilities for Data Mover (server) configuration and statistics collection.

Clients Utilities for client configurations and statistics collection.

Monitors All the System Monitors described in 4.3.

Database Database for saving performance data.

Logs Benchmarking test logs.

PkgInstall Utilities to automatically install MPFS software.

Help Online help information for using MPFS benchmark

4.5 Database

CFSPM system uses a database to save all the performance data collected from the testbed. The database allows the user to track the performance change for different releases. The database is implemented using MySQL, free database software.

Chapter 5 MPFS Performance Data Analysis

5.1 Overview

This chapter presents some MPFS performance data collected from our benchmarking testbed. The performance data collection is based on the MPFS performance metrics as defined in section 3.2, which includes throughput, response time, and scalability. We also present our detailed analysis of those performance data.

Each execution of our MPFS benchmark provides an average response time versus load level pair, and these are combined to form a MPFS performance graph, from which the overall response time of MPFS operation can be derived. Section 5.2 presents the MPFS performance graphs for both Solaris and Windows NT. The MPFS scalability is measured through gradually increasing the number of load generators and tracking the maximum aggregate throughput. Section 5.3 presents the graphs of maximum aggregate throughput versus number of load generators for both Solaris and Windows NT. Changing the operation mix percentage may provide another insight into MPFS performance. Section 5.4 presents two graphs to illustrate how the change of operation mix affects the MPFS performance from two perspectives. Since the major design goal of MPFS is to achieve better performance than NFS does, a comparison of performance between MPFS and NFS using our benchmark is presented in section 5.5.

5.2 Throughput and Response Time

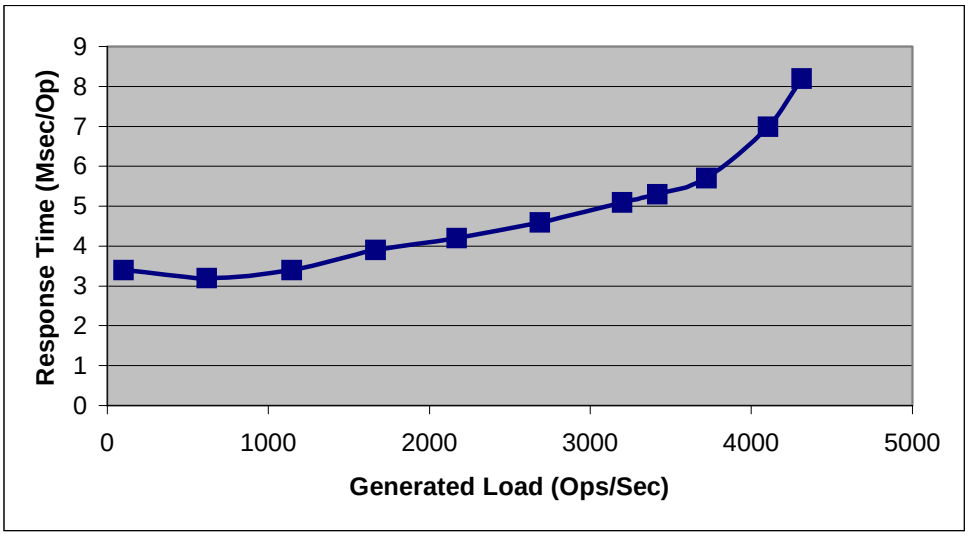

Figure 5.1Generated Load Vs. Response Time for a MPFS Benchmarking testbed with 8 Solaris Clients

Figure 5.1 shows a graph of generated load vs. response time for the MPFS benchmarking testbed with 8 Solaris clients. Our MPFS benchmark slowly increases the load level (measured in Ops/Sec) on the MPFS benchmarking testbed while measuring the system's average response time. As shown in the graph, the response time cutoff point is about 8 milliseconds and the overall response time is about 5 milliseconds. The average response time per operation is slowly increased from 3.2 milliseconds to 5.7 milliseconds as the generated load is increased from 100 Ops/Sec to 3700 Ops/Sec. Beyond 3700 Ops/Sec the average response time increases rapidly, which indicates that 3700 Ops/Sec is the optimal operating point.

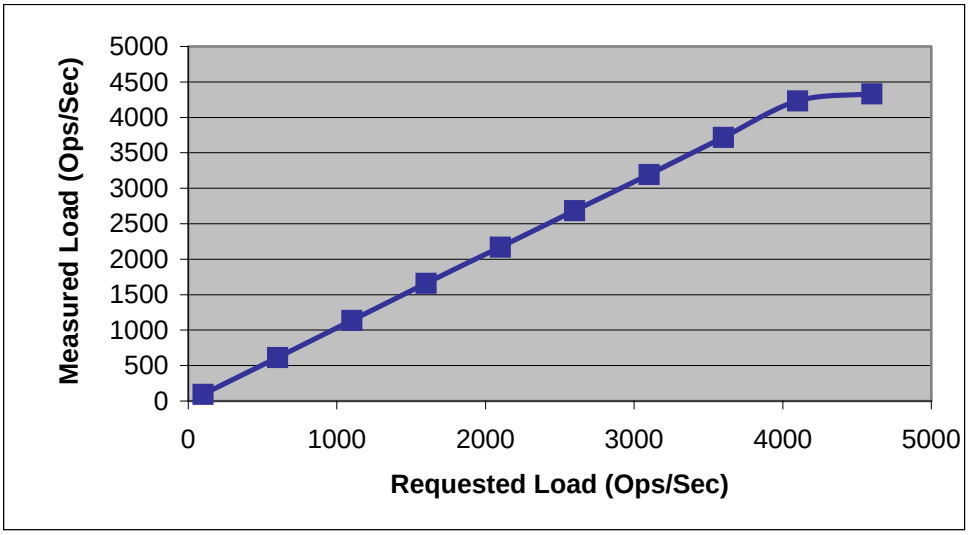

Figure 5.2 Requested Load Vs. Measured Load for a MPFS Benchmarking testbed with 8 Solaris Clients

Figure 5.2 shows the graph of the requested load vs. measured load for the MPFS benchmarking testbed with 8 Solaris clients. As shown in the graph, our MPFS benchmarking testbed can serve all requested loads before the load increases to 4200 Ops/Sec. At this point the average response time is about 8 milliseconds as shown in Figure 5.1, which is the response time cutoff point.

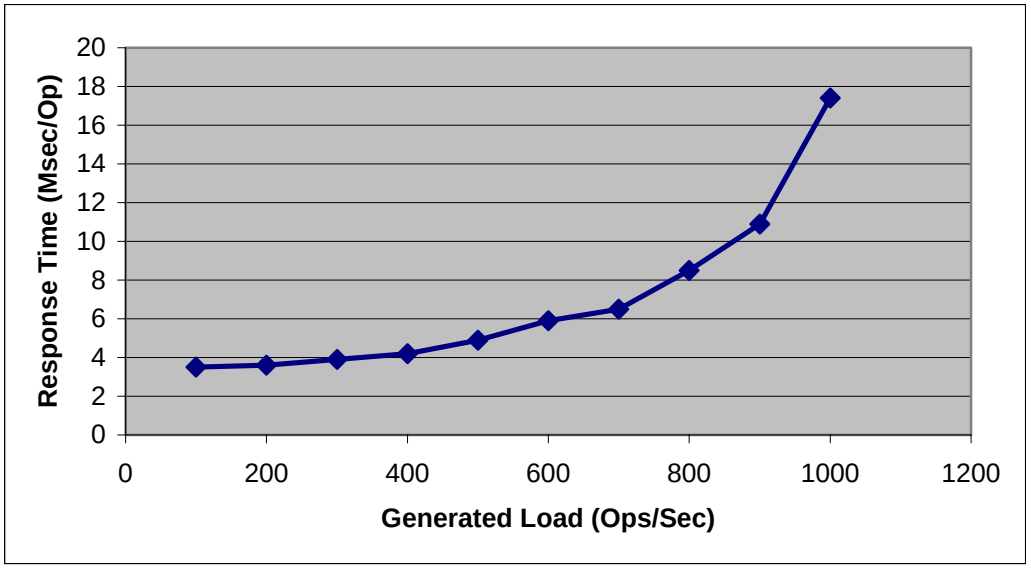

Figure 5.3 Generated Load Vs. Response Time for the MPFS Benchmarking testbed with 8 Windows NT Clients

Figure 5.3 shows a graph of generated load vs. response time for an MPFS benchmarking testbed with 8 Windows NT clients. As shown in the graph, the response time cutoff point is about 18 milliseconds and the overall response time is about 8 milliseconds. The average response time per operation slowly increases from 3.5 milliseconds to 6.5 milliseconds as the generated load is increased from 100 Ops/Sec to 700 Ops/Sec. Beyond 700 Ops/Sec the average response time increases rapidly, which indicates that 700 Ops/Sec is the optimal operating point.

5.3 Scalability

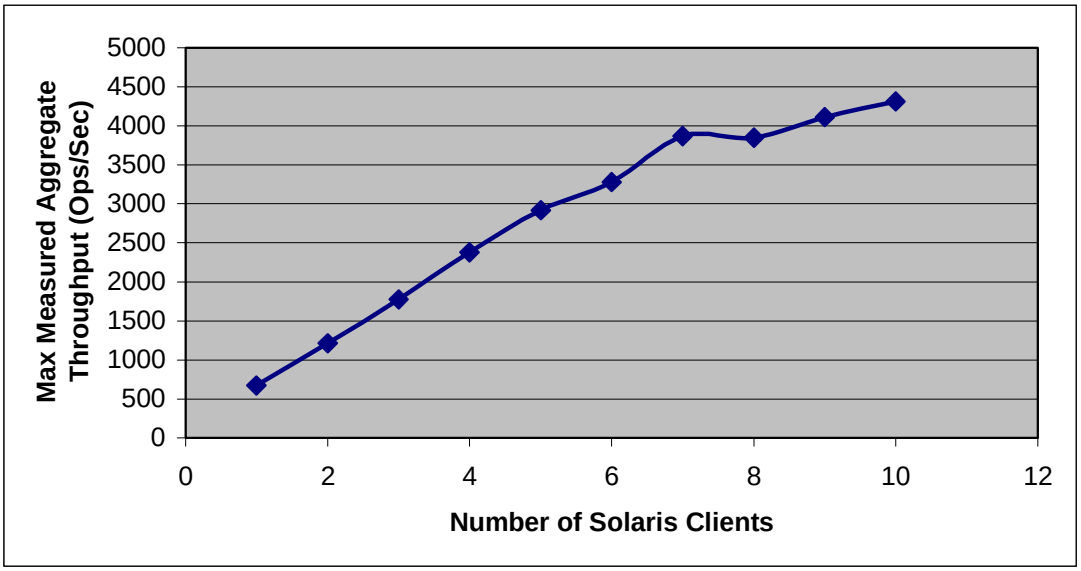

Figure 5.4 Measured Maximum Aggregate Throughput versus Number of Solaris Clients

Figure 5.4 shows a graph of measured maximum aggregate throughput versus number of Solaris clients. The measured maximum aggregate throughput represents the maximum aggregate throughput that can be achieved in our MPFS Solaris benchmarking testbed. As shown in the graph, the measured maximum aggregate throughput increases linearly as the number of clients in our testbed increases to from 1 to 7. If the number of clients in our MPFS Solaris benchmarking testbed increases beyond 7 clients, the measured maximum aggregate throughput increases only slightly.

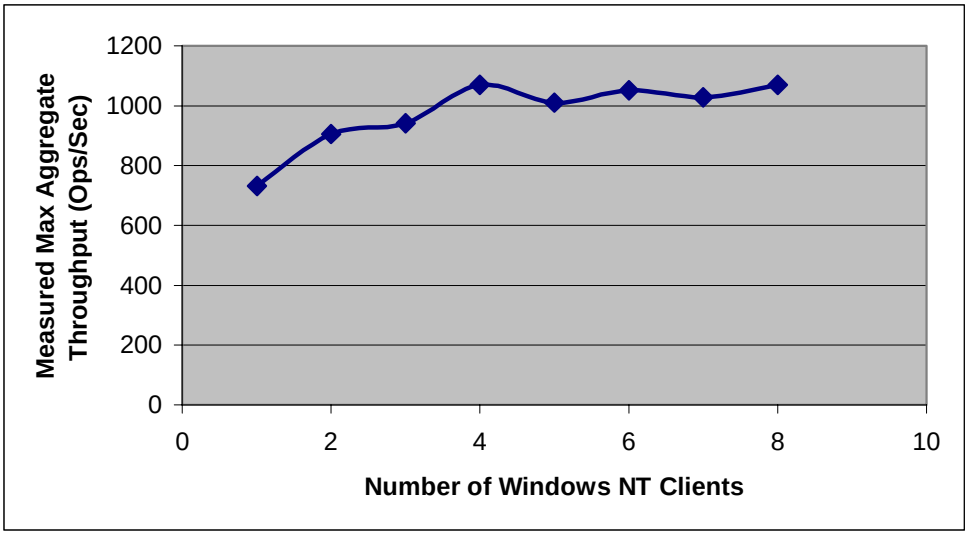

Figure 5.5 Measured Maximum Aggregate Throughput versus Number of Windows NT Clients

Figure 5.4 shows a graph of measured maximum aggregate throughput versus number of Windows NT clients. The measured maximum aggregate throughput represents the maximum aggregate throughput that can be achieved in our MPFS Windows NT benchmarking testbed. As shown in the graph, the measured maximum aggregate throughput increases from 730 Ops/Sec to 1069 Ops/Sec as the number of clients in our testbed increases from 1 to 4. If the number of clients in our MPFS Windows NT benchmarking testbed increases beyond 4 clients, the measured maximum aggregate throughput remains almost the same.

5.4 Change of the Operation Mix

The change of the operation group mix may affect the MPFS performance as shown in Figure 5.6.

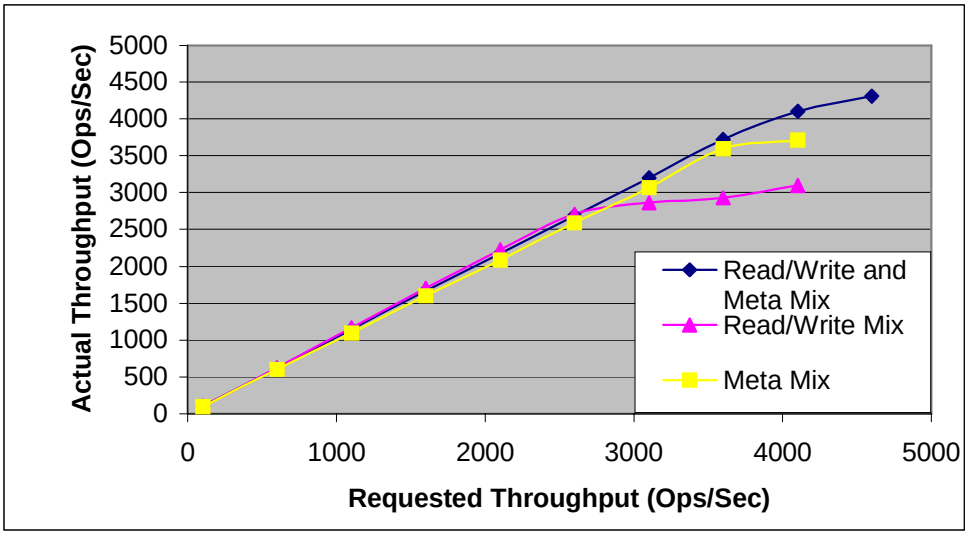

Figure 5.6 Requested Load Vs. Measured Load for different operation group mixes

Figure 5.6 depicts a graph of requested load versus measured load for three different operation group mixes. The data is collected from our MPFS Solaris benchmarking testbed of 8 Solaris clients. These three operation group mixes represent three different data traffic. Read/Write mix is I/O intensive and mainly through the I/O channel (Fiber Channel in our MPFS benchmarking testbed); Meta mix contains only metadata operations and is mainly through the network; Read/Write and Meta mix contains both I/O and metadata operations and distributes its data traffic between the network and I/O channel. The read/write mix reaches its load capacity at about 2500 Ops/Sec, which is the lowest load capacity among the three operation group mixes. The Meta mix achieves lower load capacity than Read/Write and Meta mix does. The average response time for the three operation group mixes shows the same trend, as depicted in Figure 5.7.

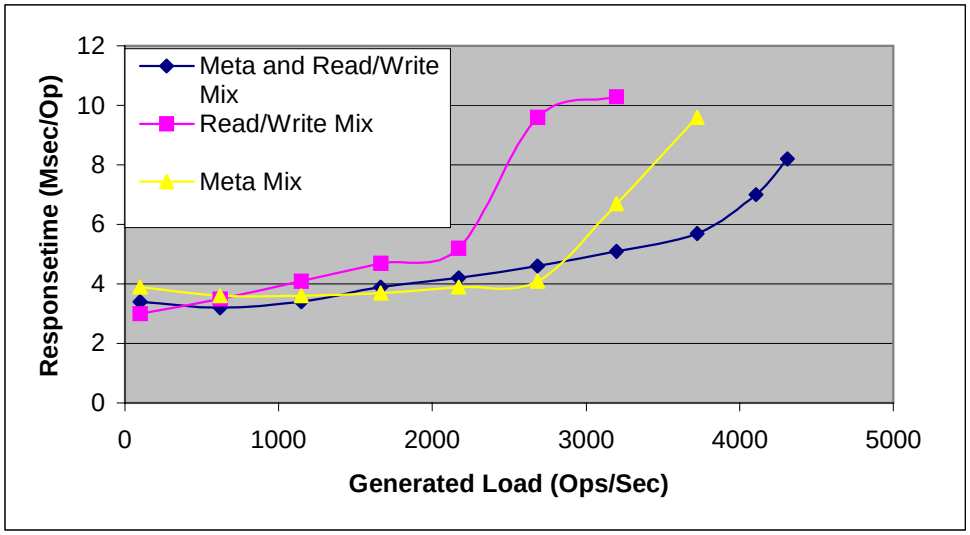

Figure 5.7 Generated Load Vs. Response Time for different operation group mixes

5.5 Comparison between MPFS and NFS

This section compares the performance between MPFS and NFS in terms of throughput, response time and scalability.

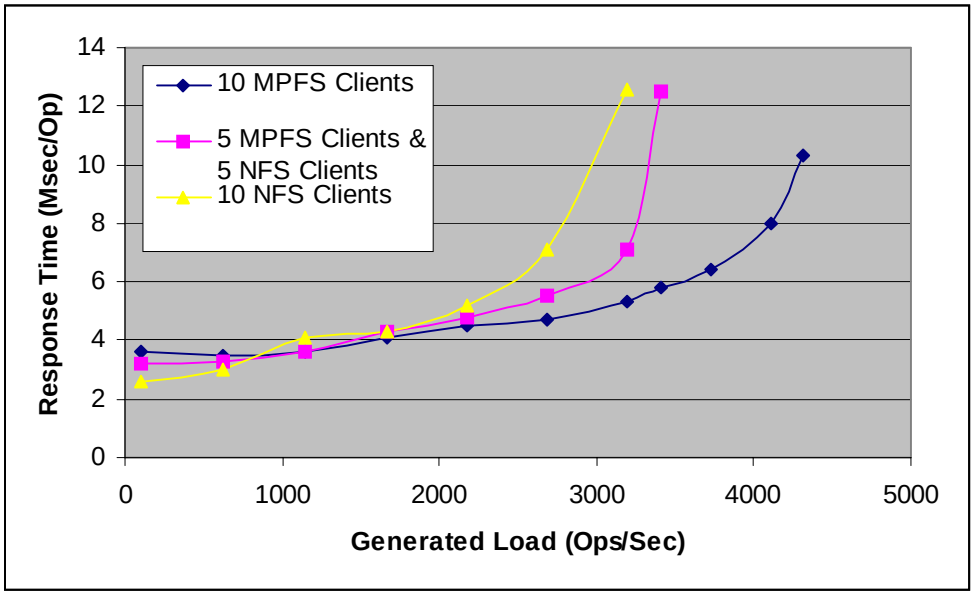

Figure 5.8 Generated Load Vs. Response Time for three different MPFS and NFS Solaris client combinations

Figure 5.8 shows the generated load versus response time for three different NFS and MPFS Solaris client combinations. It can be seen that MPFS provides better performance than NFS does over the generated loads except for at the low load levels. The performance of the mix of MPFS and NFS is better than that of NFS but worse than that of MPFS. Figure 5.9 compares the scalability between MPFS and NFS.

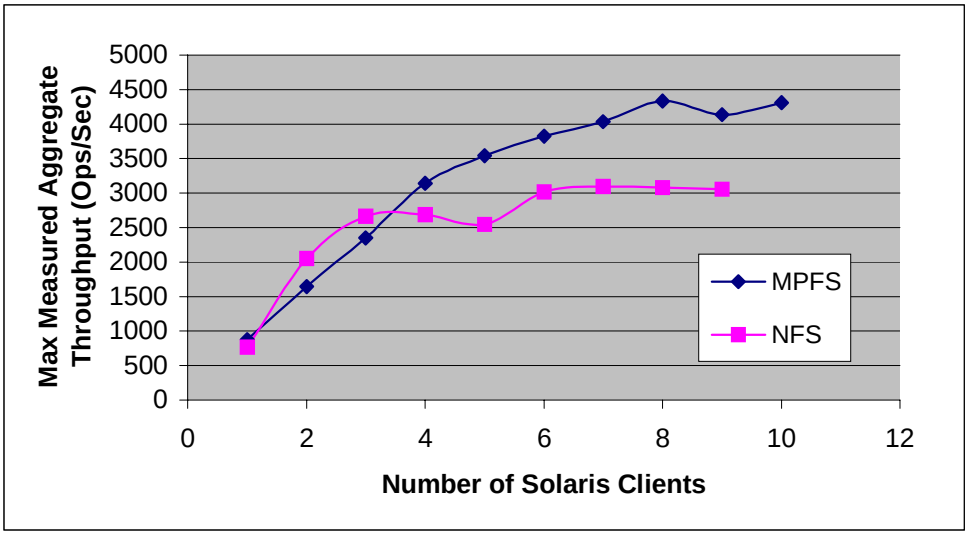

Figure 5.9 Measured Maximum Aggregate Throughputs versus Number of Solaris Clients for Comparison between MPFS and NFS

As depicted in figure 5.9, MPFS shows better scalability than does NFS after the number of Solaris clients in our testbed increases to 4. Below 4 clients, NFS shows better scalability. Figure 5.10 and Figure 5.11 further demonstrate that MPFS provides slightly better performance if the number of Solaris clients is beyond 4 but worse performance if the number of Solaris clients is below 4.

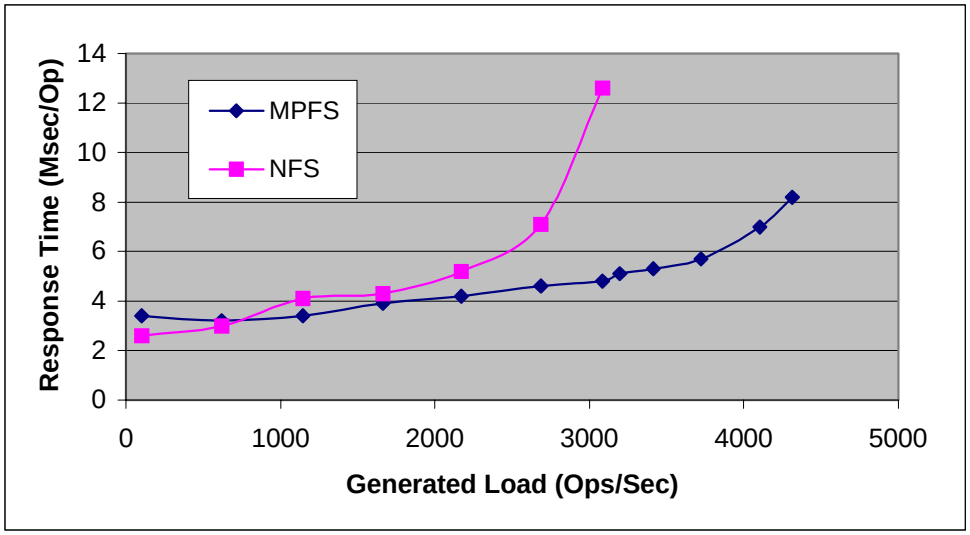

Figure 5.10 Generated Load versus Response Time for Comparison between MPFS and NFS in our MPFS Solaris benchmarking testbed with 8 clients

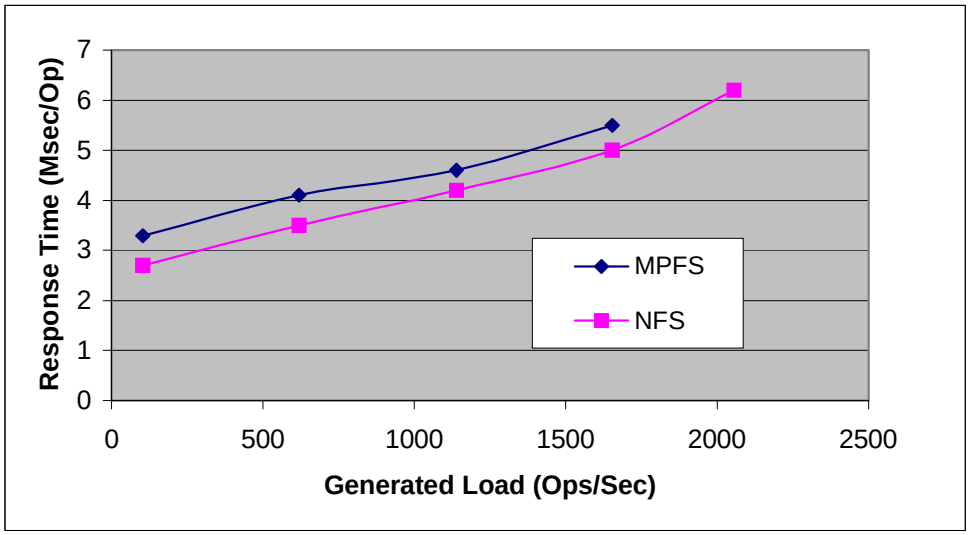

Figure 5.11Generated Load versus Response Time for Comparison between MPFS and NFS in our MPFS Solaris benchmarking testbed with 2 clients

Chapter 6 Conclusion and Future Work

6.1 Conclusion

EMC's Multiplex File System (MPFS) uses a split data-metadata architecture for distributed file systems based on shared storage, which fully exploits the special characteristics of Fiber Channel-based LANs. The hosts interact with an MPFS server for synchronization, access control, and metadata management, but do data transfers directly to and from the storage device. Most current file system benchmarks are protocol specific. They can be classified as either an NFS benchmark, such as SPEC SFS, or a CIFS benchmark, such as NetBench. Since MPFS has very different file access protocol, we can use neither of them to measure MPFS performance. MPFS allows clients to directly access storage subsystem in contrast to NFS or CIFS, in which server is the only gateway to access storage subsystem. Therefore, in order to measure MPFS performance, an effective benchmark should target both the server and the client, while most current file system only measure the server's performance. In summary, since MPFS is a totally new file access protocol, no current file system benchmarks can be used directly to measure MPFS performance, which motivates us to develop a new file system and I/O benchmark for MPFS performance evaluation.

Our benchmark consists of a number of application groups that can generate I/O intensive workloads. The application groups target representative real-world MPFS applications and mimic them at the I/O level using a set of system calls in various

combinations. The performance metrics of our benchmark include some critical performance aspects of MPFS: throughput, response time, scalability, file sharing, metadata operations and data transfer. Our benchmark can scale the file set to mimic a wide range of file sizes and distributions which MPFS applications may require. To measure some critical aspects of MPFS performance, such as read and write, our benchmark has embedded micro-benchmarks that allow users to select certain isolated components to measure. Our benchmark supports both UNIX and Windows NT clients. Based on the benchmark we propose, we develop a Web-based MPFS benchmarking testbed. Using this testbed, we collect a series of MPFS performance data.

Our MPFS benchmark uses a variety of approaches to mimic real-life MPFS applications. These approaches include allowing user to specify the application group mix for their specific applications, developing an algorithm for tuning the application group mix percentage to match the underlying NFS operation mix derived from an empirical study done by SUN [SPEC97], creating a scalable file set for our benchmarking system, using an exponentially distributed think time and simulating various real world file access patterns.

6.2 Future Work

The ideal benchmark should be relevant to a wide range of applications. It should predict a system's performance in a production environment. Currently we have no way to verify that the synthetic workload generated by our MPFS benchmark actually

represents the current or future MPFS application workloads. To improve realism, our benchmark should dynamically adjust the workload parameter according to different performance characteristics of the applications. The easiest way to characterize an application is by using trace data, but little or no trace data is available that accurately represents current or future workloads. Furthermore, as pointed out by [M99], real-world effects of using trace data make it hard to repeat experiments exactly, which reduces the benchmark's predictive value. Future work might include building up a large set of MPFS trace archives and developing a profiling model to characterize the traces. The characterization of the traces can used to generate a stochastic workload that can be executed as a benchmark.

Scalability is a major metric for MPFS performance. Currently, our benchmark uses the number of clients (load generators) to represent the scalability. This method has a clear drawback in that in a real application a load generator and a client in a real application is not a one-to-one mapping. This is due to our load-generating mechanism in which the synthetic workload generated by a load generator may mimic the workloads of more than one client. The mapping between the load generator and client in a real-world application is subject to further investigation.

The ideal benchmark should allow for fair comparisons across products. Different products have different implementations even though they use similar architectures. Since our benchmark is used for MPFS performance measurement, the workload characterization is based on the MPFS architecture; especially the operation group mix

is derived from the underlying NFS or CIFS operation mix. This workload characterization might not apply to other SAN file systems without using NFS as underlying protocol. Developing a general workload model for SAN file systems is left as future work.

References

[B90] M. Berry, Bonnie source code, http://www.textuality.com/bonnie/intro.html, 2000

[CP93a] P. Chen and D. Patterson. Storage Performance—Metrics and Benchmarks. Proceedings of the IEEE 81(8):1151-1165, Aug., 1993.

[CP93b] P. Chen and D. Patterson. A new Approach to I/O Performance Evaluation – Self-Scaling I/O Benchmarks, Predicted I/O Performance. TOCS 12 (4):308-339, 1994.

[HKMN+88] J.H. Howard, M.L. Kazar, S.G. Menees, D.A. Nichols, M. Satyanarayanan, R.N. Sidebotham and M.J. West, "Scale and Performance in a Distributed File System", ACM Transactions and Computer Systems 6, 1(February 1988), 51-81.

[KB99] Brad Kline, Distributed File System for Storage Area Networks, White Paper, http://www.adic.com/US/English/Collaterals/Documents/DistFileSystems.pdf, 2000

[KJ00] Jeffrey Katcher, PostMark: A new file system benchmark, Network Appliance, Inc., 2000

[KK99] Kent Koeninger, CXFS: A Clustered SAN File System from SGI, white paper, SGI, 1999

[KL99] B. Kline, P. Lawthers, CVFS: A Distributed File System Architecture for the Film and Video Industry, White Paper, http://www.centralvision.com/cvloindex.html, June 1999

[KR00] Robert W. Kembel, Fibre Channel, A Comprehensive Introduction, Northwest Learning Associates, Inc., 2000

[M99] Jeffrey C. Mogul, Brittle metrics in operating system research, Proceedings of 7th IEEE Workshop on Hot Topics in OperaTING systems, Rio Rico, AZ, (March, 1999), 90-95

[O98] M. T. O'Keefe "Shared (Disk) File Systems", http://www.lcse.umn.edu/GFS, 1998

[PB90] A. Park and J. C. Becker, IOStone: A synthetic file system benchmark, Computer Architecture News 18, 2 (June, 1990), 45-52

[RW93] Chris Ruemmler, John Wilkes, Unix disk access patterns, Proceedings of USENIX Winter 1993 Technical Conference, San Diego, CA, (January 25-29, 1993), 405-420

[RLA00] Drew Roselli, Jacob R. Lorch, and Thomas E. Anderson, A comparison of file system workloads, Proceedings of 2000 USENIX Technical Conference, San Diego, California (June 2000).

[SA00] SANergy File Sharing Administrators Guide, Tivoli Systems Inc, IBM, 2000

[SKS99] M. Seltzer, D. Krinsky, K. Smith, The Case for Application-Specific Benchmarking, http://www.eecs.harvard.edu/margo/papers/hotos99.html, 1999

[SPEC97] SFS 2.0 Documentation Version 1.0, Standard Performance Evaluation Corporation (SPEC), 1997.

[SRO96] S. Soltis, T. Ruwart, and M. O'Keefe, The Global File System, Fifth NASA Goddard Conference on Mass Storage Systems and Technologies, College Park, MD, September 1996.

[TPCB90] TPC Benchmark B Standard Specification, Transaction Processing Performance Council, August 1990.

[WK93] Mark Wittle and Brian Keith, LADDIS: The Next Generation in NFS File Server Benchmarking, Usenix, 1993.

[WT00] Tim Williams, NAS Protocol Extensions for NAS and SAN Data Sharing, White Paper, SNIA FILE SYSTEM WORKING GROUP, 2000

[ZD00] Understanding and Using NetBench® 6.0, ZD Inc., 2000